Memory-Efficient Mixed-Precision Implementations for Robust Explicit Model Predictive Control

MAHMOUD SALAMATI*, MPI-SWS, Germany
ROCCO SALVIA*, University of Utah, United States
EVA DARULOVA, MPI-SWS, Germany
SADEGH SOUDJANI, Newcastle University, United Kingdom
RUPAK MAJUMDAR, MPI-SWS, Germany

We propose an optimization for space-efficient implementations of explicit model-predictive controllers (MPC) for robust control of linear time-invariant (LTI) systems on embedded platforms. We obtain an explicit-form robust model-predictive controller as a solution to a multi-parametric linear programming problem. The structure of the controller is a polyhedral decomposition of the control domain, with an affine map for each domain. While explicit MPC is suited for embedded devices with low computational power, the memory requirements for such controllers can be high. We provide an optimization algorithm for a mixed-precision implementation of the controller, where the deviation of the implemented controller from the original one is within the robustness margin of the robust control problem. The core of the mixed-precision optimization is an iterative static analysis that co-designs a robust controller and a low-bitwidth approximation that is statically guaranteed to always be within the robustness margin of the original controller. We have implemented our algorithm and show on a set of benchmarks that our optimization can reduce space requirements by up to 20.9% and on average by 12.6% compared to a minimal uniform precision implementation of the original controller.

CCS Concepts: • **Computer systems organization** → Embedded software.

Additional Key Words and Phrases: model-predictive control, robustness, fixed-point arithmetic

1 INTRODUCTION

Model predictive control (MPC) is a technique to design control actions by solving finite-horizon open-loop optimal control problems at each sampling instant [38]. The result of each optimization gives a sequence of optimal control actions, only the first of which is applied to the process. The same procedure is applied in the next time instant with a shifted time horizon and a new initial state, after receiving the updated values of the process state. The optimization problem in MPC uses a dynamic model of the process, encodes all input and output (state) constraints, and optimizes a performance index. MPC has shown to be successful in a wide variety of industrial applications [35], due to its ability to systematically handle processes with many state and input variables as well as constraints on them.

The main difference between MPC and conventional control is in the nature of the function that maps the measured outputs to control actions. MPC computes such a function *online*, whereas a conventional controller pre-computes the function offline. The online computations required in MPC limits its applicability to slow processes and fast computation platforms: the sampling time has to be large enough and the platform fast enough to allow enough time for solving the

*Both authors contributed equally to this research.

This article appears as part of the ESWEEK-TECS special issue and was presented at the International Conference on Embedded Software (EMSOFT) 2019.
Authors' addresses: Mahmoud Salamati, MPI-SWS, Kaiserslauterm, Germany, msalamati@mpi-sws.org; Rocco Salvia, University of Utah, Utah, United States, rocco@cs.utah.edu; Eva Darulova, MPI-SWS, Kaiserslautern, Germany, eva@mpi-sws.org; Sadegh Soudjani, Newcastle University, Newcastle, United Kingdom, sadegh.soudjani@newcastle.ac.uk; Rupak Majumdar, MPI-SWS, Kaiserslautern, Germany, rupak@mpi-sws.org.

optimization problem and obtaining the optimal action for the next time instance. Moreover, the optimization solver needs to be certified when using MPC in safety critical applications [2].

One way to tackle these problems is through *explicit MPC* (EMPC) [2, 5], which formulates the optimization problem but computes offline a symbolic representation of the solution as a function of the state. At run-time, the solution is evaluated on the current state as in conventional control. For example, for linear time invariant models with linear constraints and quadratic costs, the optimization problem for MPC can be modeled as a quadratic program, and EMPC techniques solve the optimization problem using multi-parametric programming. The explicit solution is representable as a partition of the controller domain into a number of polyhedral regions, and an affine map for each region. Given the current state, EMPC draws the corresponding affine mapping out of a stored lookup table and evaluates the control. Explicit MPC thus expands the class of systems being controlled by MPC strategies, by taking out the need to perform massive online computations or to certify a complex optimization routine [17].

However, there are still bottlenecks in implementing EMPC on resource-constrained embedded micro-controllers. First, implementations of EMPCs on industrial micro-controllers with limited memory can suffer from large memory usage, because the solution of the optimization problem can involve many (often hundreds) of regions [17]. Since the memory consumption grows linearly with the number of regions, this can be a limiting factor in using EMPC on resource-constrained micro-controllers. Second, many low-end micro-controllers only support fixed-point arithmetic. Errors in the controller implementation are inversely proportional to the number of bits used for representing each variable [3]. An implementation of EMPC has to be *robust* to implementation errors to be able to enforce hard constraints on the states at run time [41].

In this paper, we consider the problem of implementing EMPC on low-end microcontrollers with fixed-point arithmetic in a *memory-efficient* and *robust* way. We propose an automatic controller and mixed-precision implementation co-design technique that computes mixed-precision assignments which minimize bitwidths for all variables while ensuring the resulting implementation error remains within the robustness margin.

Our proposed method iteratively solves a robust version of MPC that considers finite-precision implementation errors as disturbances in the dynamics [37]; see Figure 1. Initially, we estimate a bound $\Delta = \Delta_0$ on the implementation error and solve a min-max quadratic program for explicit robust MPC [11, 37], where the system model has an explicit disturbance bounded by Δ. The solution is represented as a set of polyhedral regions, and an affine map for each region.

Next, we consider the error in the control input arising out of a fixed-point implementation of the solution. The error has two sources. First, the fixed point implementation may pick a different polyhedral region (due to imprecision in checking membership in a polyhedron). Second, the fixed-precision implementation of the affine function will have a numerical error due to quantization. We bound both sources of error statically and automatically, by finding the maximum possible error due to incorrect region selection as well as a static bound on the error in computing the affine map. If the error is at most Δ, we know that the implementation satisfies the robustness margin in the model and we use an automated mixed-precision tuning tool to find an optimal bitwidth allocation that keeps the implementation below the error bound. If not, we increase the error bound Δ and run the loop again to find a new robust controller and find its best mixed-precision implementation.

In summary, our algorithm automatically and soundly synthesizes a suitable fixed-point mixed-precision implementation for explicit robust MPC controllers.

We have implemented our algorithm on top of Matlab's multi-parametric toolbox for robust explicit MPC [15] and the Daisy tool for multi-precision tuning and fixed-point error analysis [7].

We have applied our technique on a number of standard benchmark examples. Taking all these benchmarks into account, our algorithm finds mixed-precision implementations that save up to

20.9% memory (on average, 12.6%) in controller implementations over a minimal uniform-precision implementation (and an average saving of 46.8% over a uniform 32-bit implementation), while maintaining the correctness of the controller.

For smaller examples, the analysis takes only a few minutes of computation. We also demonstrate the scalability of our mixed-precision tuning and error bounds analysis: we show that on explicit MPC controllers with thousands of regions, our tool finishes in a few hours. In absolute terms, the memory saving corresponds to 21KB over a uniform precision implementation on our largest benchmark. A large class of EMPC applications implement the controller in processors with limited computational power and memory; typically, these processors have memory in the order of tens of kilobytes (e.g., 32 KB). Thus, 21 KB is a significant saving. Also, the memory savings do not affect the run-time performance since our controller synthesis and the corresponding mixed precision computations are all performed offline.

In summary, our contributions are:

- We provide an end-to-end automatic tuning algorithm for robust explicit MPC that ensures satisfaction of hard constraints on the state despite numerical errors;
- We design a static and scalable error analysis algorithm for piecewise affine functions to bound errors in controller implementations; and
- Through a set of standard control benchmarks, we demonstrate that our tuning algorithm can achieve up to 20% savings in memory while maintaining robustness.

2 OVERVIEW

As a simple example, consider the standard problem of designing a controller for an inverted pendulum depicted in Figure 2 (a). The goal of the controller is to keep the pendulum at the vertical position while satisfying hard constraints on the state variables and control inputs. A model of the system can be constructed using physical principles. After linearization and time discretization, the model is

$$\begin{bmatrix} \theta_{k+1} \\ \omega_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & T_s \\ \frac{T_s g}{L} & (1 - \frac{T_s b}{mL^2}) \end{bmatrix} \begin{bmatrix} \theta_k \\ \omega_k \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{T_s}{mL^2} \end{bmatrix} u_k + w_k \tag{1}$$

where θ_k, ω_k, and u_k denote respectively the angular position, angular speed, and the input torque at time kT_s with T_s being an appropriate sampling time. The disturbance $w_k \in \mathbb{R}^2$, bounded with a polyhedral set $w_k \in \mathcal{W}$, captures the modeling error due to linearization and discretization. The parameter $g = 9.81[m/s^2]$ is the gravitational acceleration, m is the ball mass, b is the rotational friction coefficient, and L is the length of the bar. Starting from an initial state (θ_0, ω_0), the control goal is to converge to the equilibrium point $\theta = 0, \omega = 0$. Additionally, we require the state constraints $\theta_k \in [-\pi, \pi]$ and $\omega_k \in [-\pi/8, \pi/8]$ to hold at all time instances.

Our overall goal is to (i) design a robust MPC controller that achieves the performance objectives including hard constraints in spite of an additional bounded disturbance Δ modeling the implementation error (both errors due to choosing a wrong region and the finite-precision computation of the control action); and at the same time (ii) to minimize the total number of bits that are required to represent an explicit controller.

Figure 1 gives a high-level overview of our proposed setup. We start by selecting an initial bound Δ_0 on the implementation error and enlarge $\mathcal{W}$ with Δ_0 as $\mathcal{W}_{\Delta_0}$, where we define

$$\mathcal{W}_\Delta := \{w_1 + w_2 \mid w_1 \in \mathcal{W}, \|w_2\| \leq \Delta\} \quad \text{for all } \Delta \geq 0. \tag{2}$$

For our example, we choose $\Delta_0 = 0.01$ and $\mathcal{W} = \{0\}$. We use Matlab to find a robust explicit MPC with disturbance set $\mathcal{W}_{\Delta_0}$. The output of robust EMPC given by Matlab decomposes the control domain into a finite number of polyhedral domains together with an affine map for each domain. At

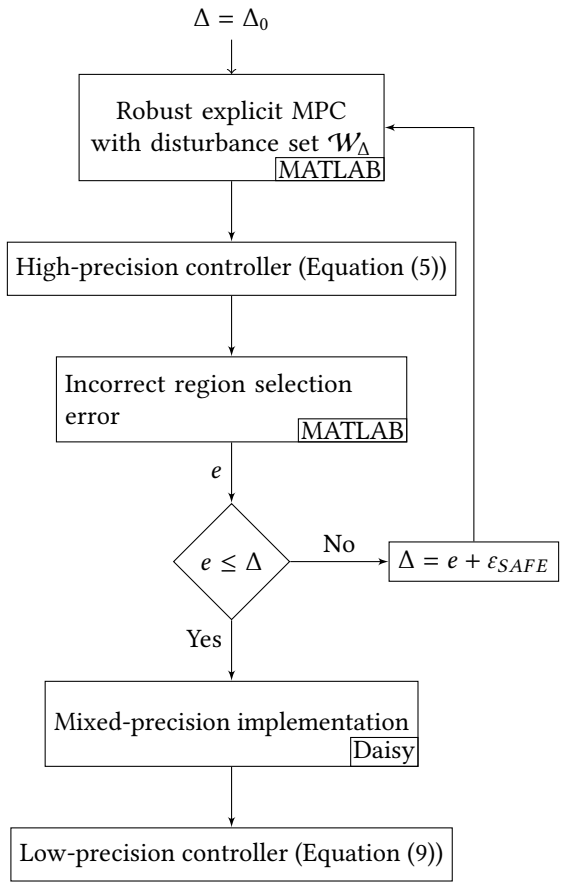


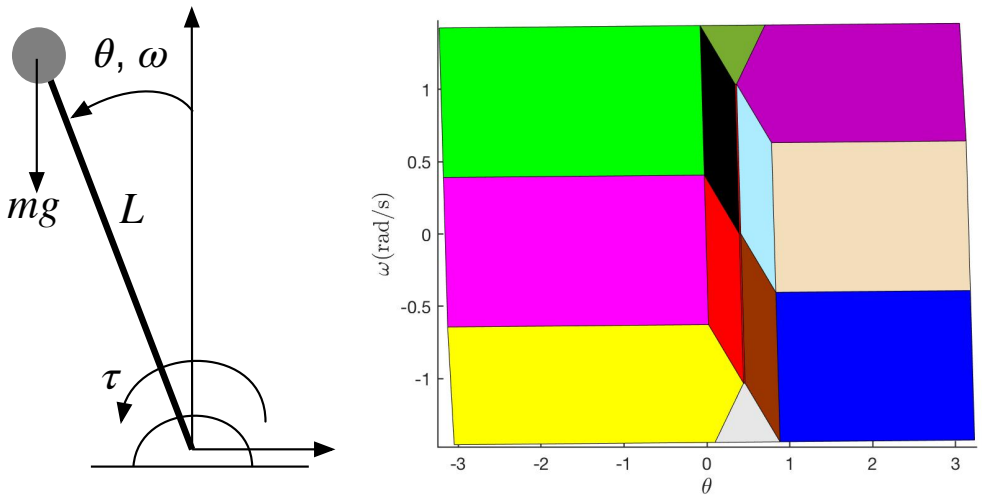
Fig. 1. Overview of the proposed memory-efficient robust EMPC control design.

each time step, the state vector (θ_k, ω_k) is read (or estimated), and based on the polyhedral domain that it belongs to, the corresponding affine map is computed as the control output.

Since the polyhedral regions are implemented in finite precision, it is possible that due to quantization errors, the polyhedral domain is selected incorrectly, and therefore a different affine map is computed for the control. We refer to the resulting error as *incorrect region selection* error. In our example, we compute an incorrect region selection error $e = 0.019$, which is larger than $\Delta_0 = 0.01$. We therefore increase the disturbance bound by a fixed amount $\varepsilon_{SAFE} = 0.031$ to $\Delta_1 = 0.05$ and enlarge the disturbance set as $\mathcal{W}_{\Delta_1}$.

Solving the robust explicit MPC problem for this enlarged disturbance set gives the new controller with 14 regions, shown in Figure 2(b). A sample region with the corresponding affine map is shown below:

$$\begin{bmatrix} -0.005 & 0.999 \\ -0.998 & -0.049 \\ 0.005 & -0.999 \\ 0.998 & 0.049 \end{bmatrix} \begin{bmatrix} \theta_k \\ \omega_k \end{bmatrix} \leq \begin{bmatrix} 0.416 \\ 3.157 \\ 0.617 \\ -0.0124 \end{bmatrix} \Rightarrow u(\begin{bmatrix} \theta_k \\ \omega_k \end{bmatrix}) = \begin{bmatrix} 0.05 & -9.67 \end{bmatrix} \begin{bmatrix} \theta_k \\ \omega_k \end{bmatrix} + 4.02$$



(a) (b)

Fig. 2. (a) Inverted pendulum; (b) 2D plot of polyhedral partitions for an EMPC for the inverted pendulum.

where the output of robust EMPC at state (θ_k, ω_k) is denoted by $u(\theta_k, \omega_k)$. The remaining mappings have a similar structure. For this new controller, the incorrect region selection error is approximately $e = 0.019$, which is now below the disturbance bound Δ_1, so that we can continue with the precision assignment.

We can implement the controller using uniform 32 bit integers, or by selecting a uniform bitwidth. These take, respectively, 13824 and 6744 bits. But we can do better. We use the precision tuning tool Daisy [7] to provide a mixed-precision implementation for all the parameters which satisfies the error bound $\Delta_1 - e = 0.05 - 0.019 = 0.031$. The mixed-precision implementation returned by Daisy requires 6084 bits. Thus, for this example, the mixed-precision implementation takes about 10% less memory compared to the smallest uniform precision implementation that respects Δ_1, and about 57% less memory than an implementation that uniformly uses 32 bits.

3 BACKGROUND

In this section, we provide relevant background on robust explicit MPC, fixed-point arithmetic and error analysis.

3.1 Robust Explicit MPC

We consider the class of linear time-invariant (LTI) systems characterized by the difference equation

$$x_{k+1} = Ax_k + Bu_k + Ew_k, \quad k = 0, 1, 2, \ldots \tag{3}$$

where $x \in \mathbb{R}^{n\times 1}$ is the state, $u \in \mathbb{R}^{m\times 1}$ is the control input and $w \in \mathbb{R}^{d\times 1}$ is the disturbance. Matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$ and $E \in \mathbb{R}^{n\times d}$ capture respectively the effects of current state, input and disturbance on the next state. We assume the disturbance w belongs to a set $\mathcal{W}_\Delta$ where $\mathcal{W}_\Delta$ is a polyhedral set defined in Equation (2). In this paper, we focus on the robust formulation of MPC which at each time step minimizes the worst-case value of an objective function with respect to the disturbances over the control inputs. We assume the objective function is quadratic with respect to

the states and inputs. We assume a linear translation on the input with the form

$$u_k = \mu x_k + v_k$$

and synthesize v_k instead of u_k. This choice reduces the conservativeness of the optimization and enlarges the set of feasible input trajectories. The matrix μ is selected such that some desired property is satisfied under suitable assumptions on the system, e.g., stability if the pair (A, B) is stabilizable.

The constrained optimization at each time step is of the form

$$\begin{aligned} J^*(x_0) = \min_{v_0,\cdots,v_{N-1}} \max_{w_0,\cdots,w_{N-1}} & \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q_F x_N \\ \text{s.t.} \quad & x_{i+1} = Ax_i + Bu_i + Ew_i, \quad \forall i \in \{0, 1, \cdots, N-1\} \\ & u_i = \mu x_i + v_i, \quad \forall i \in \{0, 1, \cdots, N-1\} \\ & u_i \in \mathcal{U}, x_i \in \mathcal{X}, \quad \forall w_i \in \mathcal{W}_\Delta, \ \forall i \in \{0, 1, \cdots, N\}, \end{aligned} \tag{4}$$

where $\mathcal{U}$ and $\mathcal{X}$ are polyhedral sets denoting the feasible sets of inputs and states. Positive definite matrices $Q \in \mathbb{R}^{n\times n}$ and $Q_F \in \mathbb{R}^{n\times n}$ indicate weights on the states. $R \in \mathbb{R}^{m\times m}$ is positive semidefinite and indicates a weight on the input in the objective function. N denotes the length of the prediction horizon.

Theorem 3.1 ([11]). *The optimization* (4) *can be translated into a multi-parametric quadratic problem which admits a closed-form solution. Furthermore, for the case that $R > 0$, the controller is $u_k = \mu x_k + \kappa(x_k)$ with $\kappa(\cdot)$ being a continuous piecewise affine (PWA) function over polyhedral regions:*

$$\kappa(x_k) = \begin{cases} F_1 x_k + G_1 & \text{if } x_k \in \mathcal{R}_1 \\ F_2 x_k + G_2 & \text{if } x_k \in \mathcal{R}_2 \\ \vdots & \\ F_P x_k + G_P & \text{if } x_k \in \mathcal{R}_P \end{cases} \tag{5}$$

where $\mathcal{R}_i$ is a polyhedral region determined by a set of linear inequalities $\mathcal{R}_i = \{x \in \mathcal{X} \mid H_i x \le K_i\}$.

The proof of this theorem can be found in [11]. The essential idea behind the theorem is to utilize the closed form $x_i = A^i x_0 + \sum_{l=0}^{i-1} A^{i-l-1} B u_l$ and transform the optimization (4) into the following quadratic program:

$$\begin{aligned} J^*(x_0) = \min_{z,\gamma} & \left[x_0^T Y x_0 + \frac{1}{2} z^T H z + \gamma \right] \\ \text{s.t.} \quad & G_m z + g_m \gamma \le W_m + S_m x_0, \\ & G_c z \le W_c + S_c x_0 \end{aligned} \tag{6}$$

where $z \in \mathbb{R}^{mN}$ and $\gamma \in \mathbb{R}$ are decision variables, and Y, H, G_m, G_c, S_m, S_c, W_m, W_c, and g_m are matrices of proper sizes that can be easily obtained from the original optimization (4) (see [11]).

The optimization (6) can be solved using multi-parametric techniques to compute the explicit form of $\kappa(\cdot)$. An efficient implementation of such computations is available in the multi-parametric toolbox of Matlab [15, 26]. Let us denote the set of states x_0 for which the optimization (6) is feasible by $\mathcal{X}_s$. Then $\mathcal{X}_s \subseteq \mathcal{X}$ and is the union of all regions $\mathcal{R}_i$:

$$\mathcal{X}_s = \bigcup_i (H_i x_k \le K_i). \tag{7}$$

```
if (H1 xk ≤ K1) then vk = F1 xk + G1
elseif (H2 xk ≤ K2) then vk = F2 xk + G2
elseif (H3 xk ≤ K3) then vk = F3 xk + G3
...
elseif (HP xk ≤ KP) then vk = FP xk + GP
return vk
```

Fig. 3. Structure of an explicit MPC controller

The implementation of the controller stores matrices F_i, G_i, H_i, K_i for all $i \in \{1, 2, \ldots, P\}$. At each time step k, the current state x_k is used to detect the polyhedral region such that $H_i x_k \leq K_i$. Then the control action $v_k = F_i x_k + G_i$ is computed and applied to the system. Several algorithms are proposed in the literature [20, 28] to find the right polyhedral region i to which the state x_k belong. The most straightforward technique is a linear search over all polyhedral regions (Fig. 3). More efficiency can be achieved by using binary search tree structures (e.g., [28]).

To keep the implementation cost down, low-end microcontrollers usually have limited computational power and memory. In general, explicit MPC is well-known for its high-speed implementation in embedded systems with low computational power. Johansen et al. [19] show that explicit MPCs with hundreds of regions can be implemented on application specific integrated circuits (ASIC) with about 20000 gates, leading to computation times in the order of $1\mu s$. It is shown in [4] that for typical problems evaluating the explicit MPC takes significantly less time in comparison to solving on-line quadratic programs. In this paper, we present an approach for minimizing the memory usage of the implementation while satisfying the hard constraints on the states, thus ensuring the controllers can be implemented in low-memory microcontrollers.

While we focus on linear time invariant systems, EMPC is also applicable to nonlinear systems by linearization around an appropriately selected equilibrium point and modeling the error in the linearization as a disturbance. Note that the rest of our analysis in the sequel (over MATLAB and Daisy) do not rely on the LTI property of the system under study and could be adjusted for other families of controllers as long as they are presented as piecewise affine functions.

3.2 Finite-Precision Implementation

We now discuss finite-precision issues in the implementation of a controller. The standard choice for low-power platforms is fixed-point arithmetic. Unlike floating-point arithmetic [16], fixed-point arithmetic can be implemented efficiently without complex hardware support using only integer operations. It requires, however, more compilation effort to determine certain operations statically at compile time that the floating-point hardware unit performs dynamically.

The main task at compile time is to select the fixed-point format for each input and intermediate value in the program. The format specifies the total word length and how many bits are available for the integer part of the number. Given ranges on program inputs, a range analysis is usually used to estimate ranges of all intermediate values in the program [24, 31], which in turn determine how many integer bits are sufficient to avoid overflow. The remaining bits of the total word length represent the fractional part of a number; more fractional bits provide more precision.

Finite precision introduces roundoff errors, which can accumulate during the course of a computation. A sound static roundoff error analysis computes a guaranteed upper bound on the error of the result by tracking worst-case errors at every operation. Given a word length for each value, several tools, including the tool Daisy which we use, automatically determine the number of integer bits needed and compute roundoff errors using a dataflow analysis [7, 9]. They track real-valued

ranges at every intermediate operation. These ranges determine the fixed-point formats and thus also the individual worst-case roundoff errors, which the analyses track separately from the ranges. To ensure guaranteed error bounds, both the ranges and errors are computed using interval [29] or affine arithmetic [10], which compute sound enclosures. Affine arithmetic tracks linear correlations between variables, so for linear expressions the computed ranges are as tight as possible (nonlinear arithmetic leads to over-approximations).

To reduce memory usage and increase efficiency, we want to choose as short word lengths as possible. This will minimize the memory footprint without influencing the run-time. Mixed-precision tuning tools automatically determine possibly different word lengths for different values [8, 25]. Compared to uniform precision (or word length), mixed-precision often leads to improved resource usage, but due to the complexity of fixed-point arithmetic and the error analysis, is challenging to do manually. Automated mixed-precision tuning tools, including Daisy, usually perform a search: they repeatedly select a candidate mixed-precision assignment (i.e. different word lengths for different values) and check whether the assignment satisfies a given error bound. The candidate assignments are chosen based on a heuristic which guides the search towards promising candidates, e.g. using delta-debugging [8] or simulated annealing [25].

In this work, we use the open-source tool Daisy which implements roundoff error analysis [7] and mixed-precision tuning [8] for fixed-point arithmetic. Fixed-point arithmetic implementations can choose different rounding modes; here we consider truncation, which is more efficient than rounding.

4 ERROR ANALYSIS

In this section, we present our error analysis assuming that a fixed word length p is given. Then in the next section, we explain our optimization algorithm which determines suitable word lengths for different variables fully automatically.

Consider the controller obtained from explicit MPC according to Theorem 3.1. Define the affine control functions associated with each polyhedral region $\mathcal{R}_i$, $i \in \{1, 2, \ldots, P\}$, as

$$v_i : \mathcal{R}_i \to \mathcal{U} \quad \text{with} \quad v_i(x) := F_i x + G_i.$$

Implementation of the controller can be affected by two main sources of error:

(i) *incorrect region selection*: instead of a correct affine function v_i, the implementation may choose an incorrect affine function v_j. This can happen either due to an analog-to-digital conversion in the measured states or due to the quantization in matrices H_i and K_i of the region $\mathcal{R}_i$; and
(ii) *approximation in the computation of the affine function*: for any selected region $\mathcal{R}_i$, the affine function v_i is evaluated using the quantized versions of F_i and G_i.

We have to ensure that the sum of these two errors remain below the bound Δ used in the design of the robust explicit MPC:

$$\max\{\|v_i - v_j\|, \forall i, j,\ i \neq j, \mathcal{R}_i \cap \mathcal{R}_j \neq \emptyset\} + \max\{err(v_i)_p,\ i = 1, 2, \ldots, P\} \leq \Delta. \tag{8}$$

The first term is the error of incorrect region selection. The maximum is taken over all neighboring regions $\mathcal{R}_i, \mathcal{R}_j$ (two regions are neighboring if their intersection is non-empty). The function norm $\|v_i - v_j\|$ is taken by maximizing over all possible values of the state x in $\mathcal{R}_i \cap \mathcal{R}_j$. The term $err(v_i)_p$ in the second part captures the approximation error in the computation of the affine function v_i when a fixed precision p is used.

4.1 Incorrect Region Selection

The first part of the error in Equation (8) captures the error due to selecting of the incorrect region. This would happen since the matrices H_i, K_i are stored in the hardware with fixed-point formats. In order to quantify the error, we define the expanded border $\bar{\mathcal{B}}_{ij}$ as the tube around the border between the regions $\mathcal{R}_i$ and $\mathcal{R}_j$:

$$\bar{\mathcal{B}}_{ij} := \left\{ x \in \mathbb{R}^n \mid \|H_i x - K_i\| \leq \varepsilon \text{ and } \|H_j x - K_j\| \leq \varepsilon \right\},$$

where ε captures the uncertainty in computing the correct region. The width of the tube ε is due to two sources of errors:

- analog-to-digital conversion $\varepsilon_{A/D}$: the error introduced by using the quantized output of the analog-to-digital converter $\hat{x}$ instead of the actual state x. We assume $\|x - \hat{x}\| \leq \varepsilon_{A/D}$.
- quantization of region bounds in memory ε_Q: the error introduced by using the quantized versions $\hat{H}_i$, $\hat{K}_i$ of H_i, K_i with p bits of precision. This is related to the boundaries of the regions.

Note that the error resulting from using the quantized versions of F_i, G_i will be discussed in Section 4.2. We define the fixed point realization of the controller in Equation (5) as

$$\hat{\kappa}(\hat{x}_k) = \begin{cases} \hat{F}_1 \hat{x}_k + \hat{G}_1 & \text{if } \hat{x}_k \in \hat{\mathcal{R}}_1 \\ \hat{F}_2 \hat{x}_k + \hat{G}_2 & \text{if } \hat{x}_k \in \hat{\mathcal{R}}_2 \\ \vdots & \\ \hat{F}_P x_k + \hat{G}_P & \text{if } \hat{x}_k \in \hat{\mathcal{R}}_P. \end{cases} \tag{9}$$

where $\hat{\mathcal{R}}_i$ is a polyhedral region determined by a set of linear inequalities $\hat{\mathcal{R}}_i = \{\hat{x} \in \mathcal{X} \mid \hat{H}_i \hat{x} \leq \hat{K}_i\}$. Let us define the sets $\mathcal{H} := \{H_i, K_i \mid i = 1, 2, \ldots, P\}$ and $\mathcal{F} := \{F_i, G_i \mid i = 1, 2, \ldots, P\}$. Similarly, we define $\hat{\mathcal{H}} := \{\hat{H}_i, \hat{K}_i \mid i = 1, 2, \ldots, P\}$ and $\hat{\mathcal{F}} := \{\hat{F}_i, \hat{G}_i \mid i = 1, 2, \ldots, P\}$. Using the triangle inequality we have:

$$\|\kappa(x) - \hat{\kappa}(\hat{x})\| \leq \|\kappa(\mathcal{H}, \mathcal{F}, x) - \kappa(\hat{\mathcal{H}}, \mathcal{F}, \hat{x})\| + \|\kappa(\hat{\mathcal{H}}, \mathcal{F}, \hat{x}) - \kappa(\hat{\mathcal{H}}, \hat{\mathcal{F}}, \hat{x})\|.$$

where $\kappa(\mathcal{H}^*, \mathcal{F}^*, x^*)$ denotes the controller defined over the sets $\mathcal{F}^*$, $\mathcal{H}^*$ and the state x^*. The second term on the right corresponds to the control approximation error and will be discussed in Section 4.2. The first term can be further decomposed as

$$\begin{aligned} &\|\kappa(\mathcal{H}, \mathcal{F}, x) - \kappa(\hat{\mathcal{H}}, \mathcal{F}, \hat{x})\| \\ &\quad \leq \|\kappa(\mathcal{H}, \mathcal{F}, x) - \kappa(\mathcal{H}, \mathcal{F}, \hat{x})\| + \|\kappa(\mathcal{H}, \mathcal{F}, \hat{x}) - \kappa(\hat{\mathcal{H}}, \mathcal{F}, \hat{x})\| \\ &\quad \leq \max_i \{\|F_i\|_2\} \|x - \hat{x}\| + \varepsilon_Q \\ &\quad \leq \max_i \{\|F_i\|_2\} \varepsilon_{A/D} + \varepsilon_Q. \end{aligned}$$

Therefore, we can compute the tube width ε as:

$$\varepsilon = \max_i \{||F_i||_2\} \varepsilon_{A/D} + \varepsilon_Q. \tag{10}$$

Analog conversion happens just before the controller receives the sensor input from the plant and is given by:

$$\varepsilon_{A/D} = \frac{V_{cc}}{2^r - 1}$$

where V_{cc} is the reference voltage of the converter (e.g. typically 5V), r is the number of bits available to quantize the analog signal, and 2^r is the resolution of the converter.

While $\varepsilon_{A/D}$ is intrinsic to the capabilities of the device, ε_Q depends on the precision used to store the boundaries:

$$\varepsilon_Q \geq \|(H_i - \hat{H}_i)\hat{x} + (K_i - \hat{K}_i)\|, \quad \forall \hat{x}, \forall i, \tag{11}$$

where $\hat{x}$ is the finite-precision value of x. That is, ε_Q bounds the distance between any hyperplane in infinite precision (H and K matrices), and its counter-part quantized by p bits ($\hat{H}$ and $\hat{K}$). This second error can be tuned providing a trade-off between accuracy and memory storage required. Therefore, the first term in Equation (8) can be computed less conservatively by maximizing only over the expanded borders:

$$\max_{x,i,j} \{\|F_i x + G_i - F_j x - G_j\|,\ x \in \bar{\mathcal{B}}_{ij}\}. \tag{12}$$

4.2 Approximate Control Output

Once a quantized state $\hat{x}$ is given and a region $\mathcal{R}_i$ is chosen, computing v_i itself introduces imprecision, because, v_i needs to be evaluated in finite-precision arithmetic:

$$err(v_i)_p \geq |(F_i - \hat{F}_i)x_k + (G_i - \hat{G}_i)|, \quad \forall x_k \in \mathcal{R}_i \cup \bar{\mathcal{B}}_{ij},\ \forall j, \tag{13}$$

where $\hat{F}$ and $\hat{G}$ represent the quantized values (in p bits) for the infinite precision values F and G.

Equation (13) is evaluated for all x_k that are inside the region $\mathcal{R}_i$, together with all the values in the tube surrounding region $\mathcal{R}_i$ (union of $\bar{\mathcal{B}}_{ij}$ for all j). In this way we compute the error also for those points that belong to a neighbor of $\mathcal{R}_i$, but because of finite-precision errors they are erroneously mapped to control action v_i.

4.3 Implementation

The incorrect region error (Equation (12)) is computed with Matlab. Since $v_i - v_j$ is also affine, and because we defined the tube as a convex region surrounding the corresponding hyperplane, it suffices to evaluate the function only at the corner points of the tube, and keep the result with the maximal magnitude. To evaluate the incorrect region selection error across the corner points, we need to first locate the corner points as well as all the regions which share those corner points. Each region R_i is represented by a set of constraints. We first extract the set of vertices of R_i using the open source Matlab library *lcon2vert* [18]. For each computed vertex v_i, we compute a n-dimensional hypercube with the edge length of 2ε. Each of the 2^n vertices of this hypercube is counted as a corner point, on which we evaluate $\|u_i - u_j\|$.

In order to find out which regions share a vertex v_i, we implemented two approaches. The first approach determines the set of neighboring regions via an exhaustive search over the set of vertices. However, we observed that this algorithm only scales to small numbers of regions. Our second approach works on the observation that all the regions R_js having v_i as their vertex can be identified if at least n of their hyperplanes cross v_i.

The error terms ε_Q and $err(v_i)_p$ are computed by Daisy. For this we encode the expressions in Equation (11) and Equation (13) respectively as straight-line arithmetic expressions and specify the constraints on the domain of x_k in the precondition. Daisy performs a dataflow analysis to determine the finite-precision roundoff errors.

While our actual synthesis algorithm (Section 5) does not compute the errors ε_Q and $err(v_i)_p$ explicitly, the error verification is used internally by Daisy to determine a suitable precision p and can also be called explicitly without the precision optimization.

Note that we use MATLAB and Daisy solely for offline computations during the compilation of the controller. At runtime, there is no additional cost.

```
1  model = input()
2  Δ = input()
3  ε_Q = input()
4  assert (Δ >= 0 and ε_Q > 0)
5  ε = max_i{||F_i||_2}ε_{A/D} + ε_Q  //width of the tube
6
7  while true:
8    F, G, H, K = design_robust_MPC(model, Δ)
9    errorRegion = incorrect_region_error(F, G, H, K, ε)
10   if Δ > errorRegion:
11     errorAct = Δ - errorRegion
12     F̂, Ĝ = precision_tuning(F, G, errorAct)
13     Ĥ, K̂ = precision_tuning(H, K, ε_Q)
14     return F̂, Ĝ, Ĥ, K̂
15   else:
16     Δ = errorRegion + ε_SAFE
```

Fig. 4. Algorithm for designing memory-efficient robust EMPC controller

5 CONTROLLER SYNTHESIS

In the previous section, we assumed a given precision p. In this section we present our algorithm for deriving suitable values of p fully automatically. Further, while Section 4 assumed a uniform finite precision, our tuning algorithm derives possibly different precisions for different control values.

Figure 4 shows a high-level view of our optimization algorithm for implementing robust MPC controllers with guaranteed error bounds. Given a state-space model of a plant (line 1), our algorithm returns four matrices $\hat{F}$, $\hat{G}$, $\hat{H}$, and $\hat{K}$, which soundly implement a robust explicit MPC controller in mixed-precision fixed-point arithmetic. Our algorithm takes two additional inputs, Δ and ε_Q (line 2, 3). The disturbance Δ provided by the user represents a starting point for the search for a robust controller. In principle, the user can set $\Delta = 0$, however note that the finite-precision implementation of the controller will always incur at least some small error. In practice we thus start the search with a slightly larger value, e.g. $\Delta = 0.1$. Our algorithm will perform a linear search and increase Δ if needed.

The third input parameter is ε_Q. In Section 4 we showed that if the user provides a precision p, then we can automatically compute ε_Q. Now, we want to *derive* a suitable precision p, however, and face an issue. In order to derive p for some expression, Daisy and any other precision tuning tool requires an error bound which should be satisfied. That is, either a precision p is given, and we can compute an error bound, or the error bound is given and we can derive a precision p. We cannot do both at the same time; the problem would be underconstrained. We solve this chicken-and-egg problem by requiring the user to provide ε_Q as part of the input.

We need to distribute the available "disturbance budget" among the two error sources: error due to selecting the wrong region, which is given by the quantization of $\hat{H}$ and $\hat{K}$, and the error due to quantization of the control action, given by $\hat{F}$, $\hat{G}$. The quantization of $\hat{H}$ and $\hat{K}$ is determined by ε_Q, thus by providing this input value, the user effectively chooses how much precision to allow for $\hat{H}$ and $\hat{K}$. Note that this precision comes at the expense of the precision of $\hat{F}$ and $\hat{G}$, i.e. the more accurately $\hat{H}$ and $\hat{K}$ are represented, the more approximate $\hat{F}$ and $\hat{G}$ need to be to fit into the disturbance budget, and vice versa. We note that one could straight-forwardly extend our

algorithm to include an outer loop which would explore different values for ε_Q. For our experiments in Section 6, we vary this value manually.

Unlike the initial Δ, the parameter ε_Q needs to be strictly positive (line 4) for mixed-precision tuning tools to work correctly. Typical values for ε_Q can range from 10^{-4} up to 10^{-2}. Line 5 computes the width of the tubes ε as shown in Equation (10), which will be used to bound the error due to selecting the wrong region ($\varepsilon_{A/D}$ is a constant parameter dependent on the specifics of the converter).

The algorithm then calls Matlab's multi-parametric toolbox to design an explicit MPC controller for the given state-space model which is robust to the initial disturbance Δ (line 8). The result is a controller represented by four matrices F, G, H, K given in high precision.

Assuming that we can implement the polyhedral region partitioning (given by matrices H, K) with an implementation error of at most ε_Q, our algorithm proceeds to compute the error due to selecting the wrong region (line 9). This computation works as explained in Section 4, Equation (12).

The algorithm then checks whether the region error remains below the disturbance bound used for the synthesis of the controller (line 10). If the error already exceeds Δ, we do not attempt to compute a finite-precision assignment, as the controller design is already infeasible. In this case, we increase Δ by a fixed (small) amount ε_{SAFE} (line 16) and design a new controller.

If the region error is smaller than Δ, we still have an error budget `errorAct` left over to implement the control actions in finite precision (line 11). For the precision assignment, we first call Daisy with the expressions for the control actions given by F and G, and the remaining error budget as the error bound (line 12). Then we call Daisy with the expressions of the region bounds given by H and K, now with the error bound ε_Q (line 13).

In both cases, Daisy determines a quantization of the input matrices (Daisy also determines the fixed-point formats of the arithmetic operations, which we ignore here). Daisy either computes a minimal uniform fixed-point precision (i.e. word length), or it returns a mixed-precision assignment, where each value and intermediate operation can potentially have a different word length. To determine the minimal uniform precision, Daisy performs a linear search. It starts from the smallest precision (1 bit) and checks whether it satisfies the error bound. If not, the precision is increased by 1 bit and the first precision that satisfies the error bound is returned. Minimal uniform precision is used as a starting point for mixed-precision tuning, which attempts to assign a lower precision to some variables. In our experiments in Section 6, we compare the two modes and show that mixed precision leads to smaller memory footprints.

Note that Daisy always returns a quantization (at least for our linear expressions). If the error bound given is very small (but larger than zero), Daisy will return a fixed-point precision with a large number bits, but will not fail.

Termination of the algorithm. From a theoretical point of view, the error due to a wrong region selection does not necessarily converge since each new designed controller may differ from the previous ones. Therefore, it is not easy to give an upper bound over the number of times that the robust MPC design needs to be repeated. However, by rewriting Equation (4), we have

$$
\begin{aligned}
&v_0(x_0,\Delta),\ldots,v_{N-1}(x_0,\Delta) = \\
&\quad \operatorname{argmin} \max_{w_0,\cdots,w_{N-1}} \sum_{i=0}^{N-1} (x_i^T Q x_i + u_i^T R u_i) + x_N^T Q_F x_N \\
&\quad \text{s.t.} \quad x_{i+1} = Ax_i + Bu_i + Ew_i, \quad \forall i \in \{0,1,\cdots,N-1\} \\
&\qquad\quad u_i = \mu x_i + v_i, \quad \forall i \in \{0,1,\cdots,N-1\} \\
&\qquad\quad u_i \in \mathcal{U}, x_i \in \mathcal{X}, \quad \forall w_i \in \mathcal{W}_\Delta,\ \forall i \in \{0,1,\cdots,N\}.
\end{aligned}
$$

Assuming the optimization has a unique solution for every Δ, one may use the results on continuity of the argmin function to compute an upper bound on the Lipschitz constant of the optimal input $\nu = \kappa(x)$ in Equation (5) that holds for all Δ. The computed bound will only depend on the dynamics of the system depicted in Equation (3) and hence can be used to evaluate the maximum error over incorrect region selection. This bound can be quite large, leading to a very conservative bound on the maximum number of iterations before convergence. From an experimental point of view, our observations show that for reasonable range of parameters, convergence was always reached after only a few iterations.

5.1 Implementation

We implemented our algorithm in a Python script which interfaces between Matlab and Daisy and which implements the high-level structure from Figure 4. The interface parses the output from Matlab and encodes the matrices F, G, H, K and the error bounds in Daisy's input format.

We set up the problem with YALMIP [26], and use Matlab's MPT toolbox [15] to compute the robust explicit MPC.

We run the precision tuning for control actions (line 12) and hyperplanes (line 13) in parallel, because the analysis of a single region or hyperplane is completely independent from the others. We currently first perform precision tuning for control actions and then for hyperplanes. These two steps could be run concurrently as well, though the impact is likely going to be minimal because the number of hyperplanes is usually an order of magnitude greater than the number of regions.

6 EXPERIMENTAL RESULTS

We evaluate a prototype implementation of the algorithm in Figure 4 on three examples.[1] For the first two, we apply the complete pipeline (design and memory optimization) which returns an end-to-end robust controller. With the third example, we evaluate the scalability of our approach when the number of regions and hyperplanes are in the order of tens of thousands.

The design of end-to-end robust controllers has been performed on a laptop with Intel i7-6700HQ CPU at 2.60GHz, with 16GB of RAM. The evaluation of the last benchmark runs on a cluster with 48 Intel Xeon v2 @ 3.00GHz cores with 1TB of RAM, of which our analysis only used 15GB.

6.1 End-to-End Robust Controller

We evaluate our complete pipeline on two benchmarks. The first one is the inverted pendulum problem depicted in Section 2, where we set $m = 0.344$ kg, $b = 0.48$ N s/m, $L = 1.703$ m and $T_s = 0.1$ s. The gain μ is selected such the $\bar{A}$ has poles at -0.1 and -0.5. Moreover, we select $N = 2$, $R = 1$ and $Q = Q_F = 100I$, where I denotes the identity matrix of proper size.

Our second benchmark is a well-known 4D example for aircraft controller design [21]. The control inputs for the aircraft 4-D model are the elevator and flaperon angles, and the attack and pitch angles are the output states that need to be regulated. The open-loop system is unstable as it has a pole with positive real part. Both control inputs are constrained between ±25 degrees. The outputs are only constrained during the first prediction horizon. You also specify scale factors for outputs. Using the gain μ, the poles of $\bar{A}$ are placed at -5, -3, -1 and -2. The robust MPC problem is solved for $N = 2$, $R = I$ and $Q = Q_F = 5I$. Note that for both of the examples, matrices R, Q and Q_F are selected such that convergence to the origin is given more weight compared to the control effort as long as constraints are satisfied.

We do not compare against the existing technique of Suardi et al. [41] which aims to reduce memory usage in explicit MPC control, as it requires the user to provide a (uniform) fixed-precision

[1] Our implementation is available online at https://github.com/rospoly/rmpc-daisy.

Table 1. Inverted Pendulum and Aircraft. Memory requirements in number of bits for storing F, G and H, K for uniform 32 bit precision (Uni32), uniform custom precision (Uni, word length chosen in parentheses) and mixed precision (Mix), for different values of Δ and ε_Q. %32vsU is the percentage of memory saved using Uni compared to the baseline Uni32, and %UvsM is the percentage of memory saved using Mix compared to Uni.

			F and G					H and K				
	Δ	ε_Q	Uni32	Uni	Mix	%32vU	%UvM	Uni32	Uni	Mix	%32vU	%UvM
pendulum	**0.30**	0.001	2688	924 (p=11)	759	65.6%	17.9%	11136	5568 (p=16)	4991	50.0%	10.4%
	0.20	0.001	2688	924 (p=11)	806	65.6%	12.8%	11136	5568 (p=16)	4992	50.0%	10.3%
	0.10	0.001	2688	1008 (p=12)	908	62.5%	9.9%	11136	5568 (p=16)	4992	50.0%	10.3%
	0.08	0.001	2688	1092 (p=13)	946	59.4%	13.4%	11136	5568 (p=16)	4992	50.0%	10.3%
	0.05	0.001	2688	1176 (p=14)	1030	56.3%	12.4%	11136	5568 (p=16)	4992	50.0%	10.3%
	0.1	**0.0006**	2688	1008 (p=12)	891	62.5%	11.6%	11136	5916 (p=17)	5261	46.9%	11.1%
	0.1	**0.0008**	2688	1008 (p=12)	901	62.5%	10.6%	11136	5568 (p=16)	5135	50.0%	7.8%
	0.1	**0.0010**	2688	1008 (p=12)	908	62.5%	9.9%	11136	5568 (p=16)	4992	50.0%	10.3%
	0.1	**0.0030**	2688	1092 (p=13)	993	59.4%	9.1%	11136	4872 (p=14)	4462	56.3%	8.4%
	0.1	**0.0050**	2688	1680 (p=20)	1527	37.5%	9.1%	11136	4872 (p=14)	4204	56.3%	13.7%
aircraft	**0.30**	0.001	9984	6864 (p=22)	6210	31.3%	9.5%	79872	64896 (p=26)	53059	18.8%	18.2%
	0.20	0.001	10368	7452 (p=23)	6725	28.1%	9.8%	82944	67392 (p=26)	55098	18.8%	18.2%
	0.10	0.001	10368	7776 (p=24)	7134	25.0%	8.3%	82944	67392 (p=26)	55098	18.8%	18.2%
	0.08	0.001	10368	7776 (p=24)	7275	25.0%	6.4%	82944	67392 (p=26)	55098	18.8%	18.2%
	0.05	0.001	10368	8424 (p=26)	7840	18.8%	6.9%	82944	67392 (p=26)	55098	18.8%	18.2%
	0.1	**0.0006**	10368	7776 (p=24)	7047	25.0%	9.4%	82944	69984 (p=27)	55705	15.6%	20.4%
	0.1	**0.0008**	10368	7776 (p=24)	7125	25.0%	8.4%	82944	69984 (p=27)	57859	15.6%	17.3%
	0.1	**0.0010**	10368	7776 (p=24)	7134	25.0%	8.3%	82944	67392 (p=26)	55098	18.8%	18.2%
	0.112	**0.0030**	10368	9720 (p=30)	9051	6.3%	6.9%	82944	64800 (p=25)	52754	21.9%	18.6%
	0.185	**0.0050**	10368	9720 (p=30)	9051	6.3%	6.9%	82944	62208 (p=24)	47877	25.0%	23.0%

Table 2. Double Integrator. N is the prediction horizon in RMPC, time gives the execution time in minutes, Regs is the number of regions of the controller with Hyps hyperplanes. Uni32 is the total number of bits when all operations are in 32 bits, Uni the minimal uniform precision required, Mix is mixed-precision, %32vU and %UvM give the improvements of uniform and mixed precisions.

				F and G					H and K				
N	time	Regs	Hyps	Uni32	Uni	Mix	%32vU	%UvM	Uni32	Uni	Mix	%32vU	%UvM
2	2	9	72	1728	810 (p=15)	628	53.1%	22.5%	13824	7776 (p=18)	7280	43.8%	6.4%
5	9	53	424	10176	5088 (p=16)	3623	50.0%	28.8%	81408	45792 (p=18)	42656	43.8%	6.8%
8	23	143	1144	27456	13728 (p=16)	9864	50.0%	28.1%	219648	123552 (p=18)	114948	43.8%	7.0%
11	47	277	2216	53184	26592 (p=16)	18980	50.0%	28.6%	425472	239328 (p=18)	222616	43.8%	7.0%
14	73	431	3446	82752	41376 (p=16)	28685	50.0%	30.7%	661632	372168 (p=18)	346020	43.8%	7.0%
17	106	621	4968	119232	59616 (p=16)	40503	50.0%	32.1%	953856	536544 (p=18)	498668	43.8%	7.1%
21	150	928	7432	178368	89184 (p=16)	59409	50.0%	33.4%	1426944	802656 (p=18)	745936	43.8%	7.1%
25	223	1299	10392	249408	124704 (p=16)	81889	50.0%	34.3%	1995264	1122336 (p=18)	1043456	43.8%	7.0%
40	314	1829	14632	351168	175584 (p=16)	113979	50.0%	35.1%	2809344	1580256 (p=18)	1469834	43.8%	7.0%

up front. Instead, we let Daisy find the minimum uniform precision needed. Additionally, we compare against a uniform 32-bit precision baseline, which in the absence of special user insight would be a reasonable safe choice.

Table 1 shows the total number of bits required to implement each controller using different precision options: uniform 32 bit precision ('Uni32'), minimal uniform precision ('Uni', chosen precision in parentheses) and mixed precision ('Mix'). We split the memory requirement into the bits required for storing F, G, H and K. We show the results for each benchmark for ten different combinations of Δ and ε_Q, varying one while keeping the other fixed. For the pendulum, the

execution time of the whole analysis is 10 minutes no matter the initial values for Δ and ε_Q, while for the aircraft it is 36 minutes.

For the inverted pendulum, minimal uniform precision saves on average 52.6% of memory compared to a uniform 32 bit baseline overall, i.e. for F, G, H and K together. Mixed-precision further reduces the memory requirement by 10.5% on average with respect to the minimal uniform baseline (and 57.5% w.r.t to uniform 32 bit baseline). Table 1 shows a more detailed breakdown of the memory requirements and savings between F, G, H and K. The memory requirements for the storage of hyperplanes H and K depends only on the size of the tubes ε_Q so that memory requirements remain constant for fixed ε_Q.

For the aircraft example, minimal uniform precision saves on average 19.3% overall w.r.t. a uniform 32 bit baseline, and mixed-precision saves an additional 17.7% w.r.t. minimal uniform precision (33.6% w.r.t to uniform 32 bit). We observe higher relative memory savings by mixed-precision for storing H and K than for the inverted pendulum example.

For the inverted pendulum MPT toolbox [15] computes a controller consisting of 14 regions with 58 2D hyperplanes in total for all choices of ε_Q and Δ. For the aircraft model, Matlab computes a robust EMPC with 27 regions and 217 4D hyperplanes for most values of ε_Q and Δ. The only exception is when $\Delta = 0.30$ and $\varepsilon_Q = 0.001$ for which we get 26 regions having 208 hyperplanes. In general, we expect that increasing Δ results in shrinking of feasible set size.

As expected, when Δ decreases, the control actions (F, G) need to be implemented more precisely and require more memory, because the space for approximation error is reduced. Similarly, when the value of ε_Q increases, the memory requirements for H and K can be relaxed.

We note that for the aircraft example, the precision for F and G is almost double with respect to the pendulum. This is because the magnitude of F and G is on the order of 10^3 while for the pendulum it is on the order of several units. Note, however, that for F and G the memory gain from minimal uniform to mixed precision (%UvM) is only slightly less than the one for the pendulum.

For the aircraft example, when $\varepsilon_Q = 0.0030$, the error due to selecting the wrong region ($max|U_i - U_j|$) exceeds the given value of $\Delta = 0.1$ and our algorithm needs to design a new controller (corresponding to the `else` branch in Figure 4). The loop converges after 5 iterations (`then` branch in our algorithm) with $\Delta = 0.112$. In each iteration, the value for Δ is increased by $\varepsilon_{SAFE} = 10^{-3}$. Thus, the algorithm reduces memory demand at the expense of slightly more disturbance for the controller. When $\varepsilon_Q = 0.0050$ the analysis converges after 4 iterations. For the other values of Δ and ε and for the pendulum example, the loop in Figure 4 is executed only once.

6.2 Scalability

The goal of the experiment in this section is to show that our algorithm works well even for the case that the controller consists of thousands of regions. This experiment is different from the end-to-end case in the sense that designing robust EMPC for the error bound Δ is replaced with designing EMPC that might not account for Δ. Given an EMPC, one can perform robustness analysis to come up with an input error bound Δ under which the performance specifications are satisfied. From the implementation point of view, this helps us to generate controllers with thousands of regions, skiping the restrictions for designing robust EMPC with longer time horizons.

The benchmark in this experiment is the double integrator, a canonical example of a second order control system. The state space description for the discrete time version of the double integrator is given by:

$$x_{k+1} = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ T_s \end{bmatrix} u_k \tag{14}$$

where, $T_s = 0.1$ is the sample time. The state and input constraints are

$$\begin{bmatrix} -5 \\ -5 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \quad -1 \leq u_k \leq 1$$

For this example, Δ and ε_Q are fixed to 0.1 and 0.001, respectively. By changing the time horizon N, we evaluate the scalability of our approach for controllers with very large number of regions, as increasing N generally results in a higher number of regions for the output controller. To compute the explicit model predictive controllers for different time horizons, we use Matlab's MPC toolbox.

The design of the controller in Matlab takes a few minutes, then controllers and hyperplanes are encoded in Daisy for finite-precision assignment. On average, the analysis of a single controller or hyperplane in Daisy requires less than two minutes and less than 500MB of memory. The finite-precision assignment is trivially parallelizable, because any controller or hyperplane can be analyzed independently. We thus run this analysis on a cluster with 48 cores, and note that the memory consumption remains below 15GB.

Table 2 shows the results of this experiment. We observe that our analysis scales up to 1829 controllers and 14632 hyperplanes in a few hours. The computed minimal uniform precision saves on average 44.5% of memory compared to a uniform 32 bit baseline overall (for F, G, H and K together). Moreover, mixed-precision reduces the memory requirement by an additional 9.3% on average with respect to the minimal uniform baseline (49.6% w.r.t. uniform 32 bit). We further observe that relative savings remain largely constant for different prediction horizons.

7 RELATED WORK

Any digital controller can be implemented using a wide range of available platforms. Large-size manufacturers use industrial digital computers called *programmable logic controllers (PLC)* for such implementations. A PLC is able to perform the computations required by a digital controller using floating-point arithmetic. Low-end applications utilize microcontrollers with limited memory capacity and computational power to keep the costs of the implementation down. We target implementations for low-end microcontrollers.

Verification of Finite-precision Controller Implementations. While control design algorithms often consider disturbance or noise, they usually assume an ideal infinite-precision implementation of the controller. Several works have explicitly considered the effects of finite-precision implementations on controller robustness.

Given a precision by the user, Suardi et al. [41] present an algorithm which iteratively designs a robust MPC controller. Similar to our loop, each iteration bounds the implementation error due to fixed-point arithmetic and if it exceeds the initial disturbance used for the design, repeats the process with an adjusted disturbance. This approach does not optimize for the precision and bounds implementation errors by reduction to an approximate LP problem. The authors further note that the iterations may diverge, i.e. a controller may not be found. Since our algorithm optimizes and adjust precision dynamically it will eventually find an implementation.

Anta et al. [3] take an existing controller and its implementation, derive safe bounds from the controller under which stability is guaranteed and verify that the implementation in fixed-point arithmetic satisfies this error bound. Similarly, Park et al. [33] takes an existing floating-point controller implementation, reconstructs the controller and verifies that roundoff errors due to finite precision remain below a user-given error bound. The LCV tool [34] additionally checks that the generated code is equivalent (within an error bound) to a Simulink block diagram model.

Unlike previous work which assumed a given implementation precision, the approach presented in this paper synthesizes both the controller and the fixed-point implementation *at the same time* and thus provides a fully automated approach.

Abate et al. [1] design safe feedback controllers with counterexample guided inductive synthesis (CEGIS). Safety verification needs to consider quantization errors in the controller and in the plant model as the algorithm is based on bounded model checking (BMC) and tracks roundoff errors with interval arithmetic. While this algorithm generates the controller and its implementation, due to limitations in BMC it only considers uniform fixed-point word length in steps of 8.

Ingole et al. [17] propose to use universal numbers (unums) instead of traditional floating-point or fixed-point arithmetic for implementing robust controllers, but without an error analysis. While unums can reduce memory footprint w.r.t. a floating-point implementation, their comparison against fixed-point arithmetic in terms of memory and performance is unclear.

Controller Synthesis. The problem of controller synthesis under safety requirements on the states has been investigated mostly for Model Predictive Control (MPC) [6] (also called receding horizon control). Researchers have investigated designing MPC controllers for satisfying safety requirements expressed as temporal logic formulas [13, 22, 23, 32, 36, 42]. The main technique is to optimize the robust satisfaction of the formula [12] (i.e., a quantitative measure of satisfaction). These works utilize MPC an an online method that requires solving at runtime often computationally expensive optimization problems. In contrast, the explicit MPC used in our work performs controller synthesis at the design time. Synthesizing digital controllers with formal guarantees while having finite-precision implementation is studied in [40] for stochastic systems.

Saha and Majumdar [39] consider memory optimization for event-driven controllers using a scratchpad to selectively load control parameters based on the current state. Our work, in contrast, is a static mapping of controller parameters.

Finite-Precision Optimization. General-purpose techniques for synthesizing fixed-point implementations of arithmetic expressions have been developed in the space of embedded systems, where resource efficiency is generally important. Some work has used dynamic analyses for estimating errors [14, 27], which, however, do not provide guarantees. Several approaches [24, 25, 30, 31] use sound error analysis techniques and perform mixed-precision optimization [25, 31], similar to the approach implemented in the tool Daisy used in this paper. An alternative way of bounding errors in fixed-point implementations of EMPC through mixed integer programming is given by Suardi et al. [41].

8 CONCLUSION

We have described an automatic technique to tune an explicit MPC controller. Our technique implements the controller in mixed-precision fixed-point arithmetic while ensuring that the resulting loss of precision does not invalidate the constraints of the original control problem. We model potential fixed-point imprecision explicitly as a disturbance term, and uses robust explicit MPC to design a controller. A static error analysis and a mixed-precision tuning tool is then used to find an efficient implementation of the controller function. The implementation maintains the numerical error to within the disturbance bounds used by the robust controller. In experiments, we show that our technique can yield significant savings in memory, ranging up to 20% in our experiments. In addition, the static analysis scales to controllers with hundreds of regions. One direction for future work would be to investigate the effect of stochastic disturbances on the proposed methodology and find a scheme to statistically reduce the memory footprint of the controller implementation.

REFERENCES

[1] A. Abate, I. Bessa, D. Cattaruzza, L.C. Cordeiro, C. David, P. Kesseli, D. Kroening, and E. Polgreen. 2017. Automated Formal Synthesis of Digital Controllers for State-Space Physical Plants. In *Computer Aided Verification (CAV)*. 462–482.

[2] A. Alessio and A. Bemporad. 2009. *A Survey on Explicit Model Predictive Control*. Springer, 345–369.

[3] A. Anta, R. Majumdar, I. Saha, and P. Tabuada. 2010. Automatic Verification of Control System Implementations. In *Proceedings of the Tenth ACM International Conference on Embedded Software (EMSOFT '10)*. ACM, New York, NY, USA, 9–18.

[4] Alberto Bemporad. 2006. Model Predictive Control Design: New Trends and Tools. In *Proceedings of the 45th IEEE Conference on Decision and Control*. IEEE. https://doi.org/10.1109/cdc.2006.377490

[5] A. Bemporad, M. Morari, V. Dua, and E.N. Pistikopoulos. 2002. The Explicit Linear Quadratic Regulator for Constrained Systems. *Automatica* 38, 1 (Jan. 2002), 3–20.

[6] Eduardo F Camacho and Carlos Bordons Alba. 2013. *Model predictive control*. Springer Science & Business Media.

[7] E. Darulova, A. Izycheva, F. Nasir, F. Ritter, H. Becker, and R. Bastian. 2018. Daisy - Framework for Analysis and Optimization of Numerical Programs. In *TACAS*.

[8] E. Darulova, S. Sharma, and E. Horn. 2018. Sound Mixed-Precision Optimization with Rewriting. In *ICCPS*.

[9] F. De Dinechin, C.Q. Lauter, and G. Melquiond. 2006. Assisted Verification of Elementary Functions Using Gappa. In *ACM Symposium on Applied Computing*.

[10] L. H. de Figueiredo and J. Stolfi. 2004. Affine Arithmetic: Concepts and Applications. *Numerical Algorithms* 37, 1-4 (2004).

[11] D. Muñoz de la Peña, T. Alamo, D.R. Ramírez, and E.F. Camacho. 2005. Min-Max Model Predictive Control as a Quadratic Program. *IFAC Proceedings Volumes* 38, 1 (2005), 263–268.

[12] A. Donzé and O. Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 92–106.

[13] S. S. Farahani, R. Majumdar, V. S. Prabhu, and S. Soudjani. 2018. Shrinking Horizon Model Predictive Control with Signal Temporal Logic Constraints under Stochastic Disturbances. *IEEE Trans. Automat. Control* (2018), 1–8.

[14] A.A. Gaffar, O. Mencer, W. Luk, and P.Y.K. Cheung. 2004. Unifying Bit-Width Optimisation for Fixed-Point and Floating-Point Designs. *FCCM* (2004).

[15] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari. 2013. Multi-Parametric Toolbox 3.0. In *2013 European Control Conference (ECC)*. 502–510. https://doi.org/10.23919/ECC.2013.6669862

[16] Computer Society IEEE. 2008. IEEE Standard for Floating-Point Arithmetic. *IEEE Std 754-2008* (2008).

[17] Deepak Ingole, Michal Kvasnica, Himeshi De Silva, and John Gustafson. 2017. Reducing Memory Footprints in Explicit Model Predictive Control using Universal Numbers. *IFAC-PapersOnLine (20th IFAC World Congress)* 50, 1 (2017), 11595 – 11600.

[18] Matt J. 2017. Analyze N-dimensional Polyhedra in terms of Vertices or (In)Equalities. https://de.mathworks.com/matlabcentral/fileexchange/30892-analyze-n-dimensional-polyhedra-in-terms-of-vertices-or-in-equalities.

[19] Tor A. Johansen, Warren Jackson, Robert Schreiber, and Petter Tondel. 2007. Hardware Synthesis of Explicit Model Predictive Controllers. *IEEE Transactions on Control Systems Technology* 15, 1 (Jan. 2007), 191–197. https://doi.org/10.1109/tcst.2006.883206

[20] C.N. Jones, P. Grieder, and S.V. Raković. 2006. A logarithmic-time solution to the point location problem for parametric linear programming. *Automatica* 42, 12 (dec 2006), 2215–2218.

[21] P. Kapasouris, M. Athans, and G. Stein. 1988. Design of feedback control systems for unstable plants with saturating actuators. *NASA STI/Recon Technical Report N* 89 (Nov. 1988).

[22] Sertac Karaman, Ricardo G. Sanfelice, and Emilio Frazzoli. 2008. Optimal control of Mixed Logical Dynamical systems with Linear Temporal Logic specifications. In *Proc. of CDC 2008: the 47th IEEE Conference on Decision and Control*. IEEE, 2117–2122.

[23] E.S. Kim, S. Sadraddini, C. Belta, M. Arcak, and S.A. Seshia. 2017. Dynamic contracts for distributed temporal logic control of traffic networks. In *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE, 3640–3645.

[24] A. B. Kinsman and N. Nicolici. 2009. Finite Precision Bit-Width Allocation using SAT-Modulo Theory. In *DATE*.

[25] D. U. Lee, A. A. Gaffar, R. C. C. Cheung, O. Mencer, W. Luk, and G. A. Constantinides. 2006. Accuracy-Guaranteed Bit-Width Optimization. *Trans. Comp.-Aided Des. Integ. Cir. Sys.* 25, 10 (2006).

[26] J. Lofberg. 2004. YALMIP : a toolbox for modeling and optimization in MATLAB. In *2004 IEEE International Conference on Robotics and Automation (IEEE Cat. No.04CH37508)*. 284–289. https://doi.org/10.1109/CACSD.2004.1393890

[27] A. Mallik, D. Sinha, P. Banerjee, and H. Zhou. 2007. Low-Power Optimization by Smart Bit-Width Allocation in a SystemC-Based ASIC Design Environment. *IEEE Trans. on CAD of Integ. Cir. and Sys.* (2007).

[28] M. Mönnigmann and M. Kastsian. 2011. Fast explicit MPC with multiway trees. *IFAC Proceedings Volumes* 44, 1 (jan 2011), 1356–1361.

[29] R.E. Moore. 1966. *Interval Analysis*. Prentice-Hall.

[30] W. G. Osborne, R. C. C. Cheung, J. Coutinho, W. Luk, and O. Mencer. 2007. Automatic Accuracy-Guaranteed Bit-Width Optimization for Fixed and Floating-Point Systems. In *FPL*.

[31] Yu Pang, Katarzyna Radecka, and Zeljko Zilic. 2011. An Efficient Hybrid Engine to Perform Range Analysis and Allocate Integer Bit-widths for Arithmetic Circuits. In *ASPDAC*.

[32] Y.V. Pant, H. Abbas, and R. Mangharam. 2017. Smooth operator: Control using the smooth robustness of temporal logic. In *Control Technology and Applications (CCTA), 2017 IEEE Conference on*. IEEE, 1235–1240.

[33] J. Park, M. Pajic, O. Sokolsky, and I. Lee. 2017. Automatic Verification of Finite Precision Implementations of Linear Controllers. In *Tools and Algorithms for the Construction and Analysis of Systems TACAS, Uppsala, Sweden, April 22-29*. 153–169.

[34] Junkil Park, Miroslav Pajic, Oleg Sokolsky, and Insup Lee. 2019. LCV: A Verification Tool for Linear Controller Software. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 213–225.

[35] S.J. Qin and T.A. Badgwell. 2003. A survey of industrial model predictive control technology. *Control engineering practice* 11(7) (2003), 733–764.

[36] V. Raman, A. Donzé, M. Maasoumy, R.M Murray, A. Sangiovanni-Vincentelli, and S.A. Seshia. 2014. Model predictive control with signal temporal logic specifications. In *Decision and Control (CDC), 2014 IEEE 53rd Annual Conference on*. IEEE, 81–87.

[37] D.R. Ramirez and E.F. Camacho. 2006. Piecewise affinity of min-max MPC with bounded additive uncertainties and a quadratic criterion. *Automatica* 42(2) (2006), 295–302.

[38] James B. Rawlings, David Q. Mayne, and Moritz M. Diehl. 2017. *Model Predictive Control: Theory, Computation, and Design* (2 ed.). Nob Hill Publishing.

[39] I. Saha and R. Majumdar. 2012. Trigger memoization in self-triggered control. In *EMSOFT 2012*. ACM, 103–112.

[40] Fedor Shmarov, Sadegh Soudjani, Nicola Paoletti, Ezio Bartocci, Shan Lin, Scott A. Smolka, and Paolo Zuliani. 2019. Automated Synthesis of Safe Digital Controllers for Sampled-Data Stochastic Nonlinear Systems. *CoRR* abs/1901.03315 (2019). arXiv:1901.03315

[41] A. Suardi, S. Longo, E.C. Kerrigan, and G.A. Constantinides. 2016. Explicit MPC: Hard constraint satisfaction under low precision arithmetic. *Control Engineering Practice* 47 (2016), 60 – 69.

[42] T. Wongpiromsarn, U. Topcu, and R.M. Murray. 2012. Receding Horizon Temporal Logic Planning. *IEEE Trans. Automat. Contr.* 57, 11 (2012), 2817–2830.